“Lean and Efficient Software: Whole-Program Optimization of Executables”

Project Summary Report #2
(Report Period: 9/30/2014 to 12/31/2014)

Date of Publication: February 19, 2015

Sponsored by Office of Naval Research (ONR)

Contract No. N00014-14-C-0037
Effective Date of Contract: 06/30/2014

Technical Monitor: Sukarno Mertoguno (Code: 311)
Contracting Officer: Casey Ross

Submitted by:

Principal Investigator: Thomas Johnson
531 Esty Street
Ithaca, NY 14850-4201
(607) 273-7340 x. 134
tjohnson@grammatech.com

Financial Data Contact:
Krisztina Nagy
T: (607) 273-7340 x.117
F: (607) 273-8752
knagy@grammatech.com

Administrative Contact:
Derek Burrows
T: (607) 273-7340 x.113
F: (607) 273-8752
dburrows@grammatech.com

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
19 FEB 2015		**30-09-2014 to 31-12-2015**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Lean and Efficient Software: Whole-Program Optimization of Executables	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **GrammaTech,531 Esty Street,Ithaca,NY,14850-4201**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**7**	

1 Financial Summary

Contract Effective Date	06/30/2014
Contract End Date	06/30/2016
Reporting Period	09/30/2014 – 12/31/2014
Total Contract Amount	$602,165
Incurred Costs this Period	$140.239
Incurred Costs to Date	$225,941
Est. Cost to Completion	$376,224

2 Project Overview

Background:

Current requirements for critical and embedded infrastructures call for significant increases in both the performance and the energy efficiency of computer systems. Needed performance increases cannot be expected to come from Moore's Law, as the speed of a single processor core reached a practical limit at ~4GHz; recent performance advances in microprocessors have come from increasing the number of cores on a single chip. However, to take advantage of multiple cores, software must be highly parallelizable, which is rarely the case. Thus, hardware improvements alone will not provide the desired performance improvements and it is imperative to address software efficiency as well.

Existing software-engineering practices target primarily the productivity of software developers rather than the efficiency of the resulting software. As a result, modern software is rarely written entirely from scratch—rather it is assembled from a number of third-party or "home-grown" components and libraries. These components and libraries are developed to be generic to facilitate reuse by many different clients. Many components and libraries, themselves, integrate additional lower-level components and libraries. Many levels of library interfaces—where some libraries are dynamically linked and some are provided in binary form only—significantly limit opportunities for whole-program compiler optimization. As a result, modern software ends up bloated and inefficient. Code bloat slows application loading, reduces available memory, and makes software less robust and more vulnerable. At the same time, modular architecture, dynamic loading, and the absence of source code for commercial third-party components make it hopeless to expect existing tools (compilers and linkers) to excel at optimizing software at build time.

The opportunity:

Our objective in this project is to substantially improve the performance, size, and robustness of binary executables by using static and dynamic binary program analysis techniques to perform whole-program optimization directly on compiled programs: specializing library subroutines, removing redundant argument checking and interface layers, eliminating dead code, and improving computational efficiency. In particular, we will apply specialization and partial evaluation technology, integrating the new technology with the techniques developed during the previous contract effort. We expect the optimizations to be applied at or

immediately prior to deployment of software, giving our tool an opportunity to tailor the optimized software to its target platform. Today, machine-code analysis and binary-rewriting techniques have reached a sufficient maturity level to make whole-program, machine-code optimization feasible. Thus, we believe there is now a great opportunity to design tools that will revolutionize the software development industry.

Work items:

We expect to develop algorithms and heuristics to accomplish the goals stated above. We will embed our work in a prototype tool that will serve as our experimental and testing platform. Because "Lean and Efficient Software: Whole-Program Optimization of Executables" is a rather long title, we will refer to the project as *Layer Collapsing* and the prototype tool as ***Laci*** (for **LA**yer **C**ollapsing **I**nfrastructure).

The specific work items for the base contract period are listed below:

1. **Investigate specialization opportunities.** The contractor will design and implement limit studies that will help focus the search for fruitful applications of partial evaluation and set goals for attainable improvements.
2. **Transfer UW technology.** The contractor will transfer program-specialization or partial-evaluation technology from the University of Wisconsin and integrate it into the contractor's tool chain.
3. **Improve and extend UW technology.** The contractor will improve the robustness and scalability of the transferred technology, and complete partially implemented components and functionality.
4. **Improve and extend IR construction and rewriting.** The contractor will improve intermediate-representation construction and rewriting infrastructure as needed to demonstrate functionality on the primary test subjects.
5. **Develop and maintain test infrastructure.** The contractor will create an extensive suite of test applications, and will maintain and extend it as necessary. The contractor will also implement validation and measurement functionality that will enable tracking the robustness and benefits of program transformations.
6. **Investigate security implications.** As time permits, the contractor will study the effect of different instruction-generation mechanisms, such as peephole superoptimization, on security. As time permits, the contractor will also study whether polyvariant specialization enables (i) the creation of finer security-relevant models of program behavior and (ii) more accurate or efficient enforcement of security policies. If earlier tasks that are essential in completing a functional prototype require more effort, we propose to shift this task to the option period, with the possible adjustments of lower effort on either or both of the first two option-period tasks.

7. **Produce deliverables and attend required meetings.** The contractor will produce technical documentation in the form of reports and a working software prototype. The contractor will attend meetings requested by the program monitor.

3 Accomplishments during the reporting period

This report covers the second quarter of the base contract period. During this quarter, we focused our efforts on completing the switch to the sieve-style rewriting that GrammaTech's ADAPT project has developed. We're happy to report that by the end of the quarter, LACI is now able to successfully rewrite statically linked executables with the new method.

In addition, we invested some effort in refining how jump tables are handled in CodeSurfer's IR recovery phase. Accurately modeling jump tables in the IR will enable LACI's transforms to make more aggressive modifications and will also make LACI's rewriting process more robust.

We continue to track the progress that University of Wisconsin (UW) is making on partial evaluation and instruction synthesis. Our expectation is that it will be mature enough in the next quarter to begin transitioning to GrammaTech in order to leverage in LACI.

In the next quarter, we plan to work on two fronts: 1) expanding LACI's new rewriting framework to support dynamically linked libraries, and 2) transitioning UW's partial evaluation and instruction synthesis technology.

The following sections provide details on these accomplishments.

3.1 Making Rewriting More Robust

As described in the report for the previous quarter, we have been converting LACI's infrastructure to leverage a new style of rewriting inspired by the REINS rewriting system. This new style was first developed under a related project at GrammaTech that is developing ADAPT, a tool for patching vulnerabilities in software. The new style uses a concept called a "sieve" to handle indirect control flow. Thus, we've taken to referring to this approach as sieve-style rewriting.

In the previous quarter, we had made progress on implementing an enhancement to CodeSurfer/SWYX to support conservative disassembly, a requirement for sieve-style rewriting. During this reporting period, we continued to fine-tune this component.

We also invested substantial effort in reworking the reassembly and linking toolchain on the backend of the rewriting process to support sieve-style rewriting. The challenge here is to retain the original program's byte signature in its original location while adding the rewritten code. The rewritten code must perform data references to the original program image, but control-flow transfers must be redirected to remain within the rewritten code portion of the new program image.

While we still have some bugs to iron out, the new rewriting system appears to function correctly on small, statically linked examples. A task moving forward will be to expand our testing to larger example.

In addition, we have a remaining challenge to support dynamically linked libraries. The challenge here is that the sieve-style rewriting does not statically alter the values of code pointers that the rewritten program manipulates. Code pointers retain the values they had in the original, unmodified program. To perform indirect jumps, the sieve-style rewriting performs a dynamic translation step to convert the values of code pointers to the appropriate new values in the rewritten program. Interfacing with dynamically linked libraries that do not support this translation step may lead to execution errors any time a code pointer crosses the boundary between the rewritten executable and the library.

There are a couple of approaches we've begun exploring to deal with this. One option is to intercept any code pointers and translate them before being passed to the library. This may be difficult to accomplish in programs that pass code pointers to libraries in complex data structures. Another approach is to rewrite the dynamic libraries a program uses in addition to the program itself. This would have the benefit of providing confidence in the rewriting system. However, it would incur maintenance overhead for users who want to update their system libraries, say. Finally, one of LACI's transformations is to translate dynamic libraries to static libraries. This enables further optimization and customization to occur, but it will also address the rewriting challenge, as the executable no longer leverages dynamic libraries.

In the end, it's likely a combination of all three approaches may work best. Our plan is to focus on the third option (leveraging LACI's dynamic-to-static library conversion) as a first step, since that is simplest to get working under the current contract. We plan to focus on this in the early part of the next quarter.

3.2 Evaluation of UW Technology

As mentioned in the previous report, we have been tracking UW's progress on developing partial evaluation and instruction synthesis. We have not yet transitioned the technology to GrammaTech. We currently expect this to occur in the next reporting period.

We have not made further progress on working with UW's specialization slicing. A key facet of the slicing is that it is overly sensitive to use of the stack pointer. Because many instructions access the stack to read or write local variables, slices end up being unreasonably large (and thus not much dead code can be identified). Our understanding of UW's work on partial evaluation and instruction synthesis is that these new capabilities may help alleviate this issue for special slicing. Thus, we've put further investigation on hold until the newer work is available.

4 Goals for the next reporting period

In the next reporting period we expect to complete the following:

- Begin integrating LACI's library conversion transformation from Phase I to support the sieve-style rewriting, enabling the rewriting to support dynamic libraries.
- Begin transitioning UW's work on partial evaluation and instruction synthesis.

5 Milestones

Interim results on multi-month tasks will be reported in the quarterly progress reports.

Milestone	Planned Start date	Planned Delivery/ Completion Date	Actual Delivery/ Completion Date
Kickoff Mtg		9/4/2014	9/4/2014
Transition Specialization Slicing	7/2014	12/2014	
Robustness & Reliability of IR & Rewriting	7/2014	12/2014	12/2014 – statically linked exes
First Quarterly Report		9/30/2014	11/21/14
Transition Partial Evaluation and Instruction Synthesis	12/2014	5/2015	
Second Quarterly Report		12/30/2014	2/19/15
Third Quarterly Report		3/30/2014	
Evaluation	4/2015	6/2015	
Final Report		6/30/2014	

6 Issues requiring Government attention

None.